Remarks

Status of application

Claims 1-61 were examined and stand rejected in view of prior art as well as for technical reasons. The distinctions between Applicant's invention and the cited prior art references are discussed in detail in the following remarks. In view of the amendments made herein and the below remarks, reexamination and reconsideration are respectfully requested.

The invention

Applicant's invention comprises an attribute-based component programming system and methodology for object-oriented languages. Applicant's solution provides a technique that is based on attributes, which comprise language constructs that a developer can use to add metadata to code elements (e.g., assemblies, modules, members, types, return values, and parameters) so as to dynamically generate code to extend program functionality. Applicant's invention provides a flexible solution in which attributes can be defined internally or externally to the application source code, or using a combination of both internal and external definitions. Externally defined attributes are derived from properties in a configuration repository. Properties are defined by simple sets of name-value pairs, which are easier to understand and use than XML deployment descriptors and aspect definition syntaxes. Attributes are also easier to understand and use than the "aspects" utilized in aspect-oriented programming (AOP).

The attributes provided by Applicant's invention provide a flexible and extensible way of modifying the behavior of a program or a class or a method of a program. Applicant's solution supports user-defined attributes, including both code generation and runtime metadata access for user-defined declarative behaviors. Prior techniques are generally inflexible in this regard and typically offer only one option (or none). These attributes comprise "active" metadata used to generate other code for inclusion in a program class or subclass. The solution supports either static and dynamic code generation, or a combination of both. Static generation allows fast application start up times, whereas dynamic generation allows for reconfiguration.

<u>General</u>

A. Objections to the Specification

The Examiner objected to the disclosure as containing embedded hyperlinks and indicated embedded hyperlinks were found at paragraphs [0034] and [0036] of Applicant's specification. Although Applicant respectfully disagrees that that the referenced paragraphs include any hyperlinks or browser-executable code contrary to MPEP §608.01 as they do not include a URL placed between the symbols "< >" or "http://" followed by a URL address, Applicant has amended its specification so as to overcome the objection.

In accordance with the request of the Examiner, Applicant has amended the specification to include references to the Java trademark when such mark is first referenced in Applicant's specification. However, Applicant takes no position as to the validity or ownership of any such trademark or whether any such referenced trademarks serve to identify any particular products.

The Examiner has also objected to the Specification as failing to provide proper antecedent basis for the subject matter of Claim 5. As to these claim limitations, Applicant respectfully disagrees with the Examiner as Applicants specification clearly describes that components may be defined with abstract classes (see e.g., Applicant's specification, paragraph [0067]; see also paragraphs [0072]-[0073]) and includes a specific example of a component class being declared as abstract (see e.g., Applicant's specification, paragraphs [0114]-[0116]. Although Applicant believes that the original claim 5 is supported in Applicant's specification, Applicant has amended claim 5 to include claim limitations of a component class declared as abstract, thereby overcoming the Examiner's objection.

B. Section 101 Rejection

Claims 25-46 stand rejected under 35 U.S.C. 101 on the basis of non-statutory subject matter. Although Applicant respectfully believes that the Examiner has incorrectly construed Applicant's specification as stating that the elements of Applicant's invention can only be implemented in software, Applicant has amended claim 26 to include limitations of a computer having a processor and memory. These limitations are supported in Applicant's specification. For example, paragraph [0043] of Applicant's specification states as follows: "...the corresponding apparatus element may be configured in hardware, software, firmware or combinations thereof" (Applicant's specification, paragraph [0033], emphasis added). Applicant's specification also describes in detail a computer hardware and software environment in which Applicant's invention may be implemented (Applicant's specification, paragraphs [0044]-[0054]; also see generally, Figs. 1 and 2). Accordingly, as Applicant's claims 26-45 define a useful machine or item of manufacture in terms of a hardware or hardware and software combination, Applicant respectfully believes that it defines a statutory product.

The Examiner has also rejected claim 25 on the same basis. However claim 25 is dependent upon and incorporates the limitations of Applicant's claim 1, which the Examiner has not rejected as non-statutory. Accordingly, Applicant respectfully believes that the Section 101 rejection of claim 25 is improper and should be withdrawn.

C. Claim Objections

The Examiner has objected to Claims 24, 25, 60 and 61 as being of improper dependent form. Applicant has amended claims 24, 25, 60 and 61, thereby overcoming the objection.

Prior art rejections

A. Section 102 rejection:

Claims 1-5, 15-19, 21-26, 28-30, 33, 40-43, and 45-46 stand rejected under 35 U.S.C. 102(b) as being anticipated by US Patent 6,182,277 to DeGroot et al (hereinafter "DeGroot"). The Examiner's rejection of Applicant's claim 1 as follows is representative of the Examiner's rejection of these claims as anticipated by DeGroot:

As per claim 1,
DeGroot et al. disclose
- adding a component object to a program class of the program to create a component (c4: 40-43, " ...method of an object... " A method is component in a class and is therefore added.);
- defining at least one attribute specifying declaratively behavior to be added to the program (c4: 40-53, " ...declarative techniques...that define object behavior_.");

> - associating said at least one attribute with the component (c4 : 1-5, " ...associate the declarative statements to ... on the object ... "); and
> - in response to instantiation of the component at runtime, generating a subclass based on the program class and said at least one attribute, the subclass including dynamically generated program code based on said at least one attribute(c2: 14-25, "_.subclassing technique permit ...to generate a new method on a subclass .. "; c8: 6-8, " ...may be added at run time ... "; c2:40-42, "…the subclassing technique requires re-compiling the code... ").

Under Section 102, a claim is anticipated only if each and every element as set forth in the claim is found, either expressly or inherently described, in the single prior art reference. (See, e.g., MPEP Section 2131.) As will be shown below, DeGroot fails to teach each and every element set forth in Applicant's claims 1-5, 15-19, 21-26, 28-30, 33, 40-43, and 45-46 (as well as other claims), and therefore fails to establish anticipation of the claimed invention under Section 102.

The Examiner equates Applicant's attribute-based component programming system and methodology to DeGroot's declarative programming techniques for object oriented environments. DeGroot's declarative programming technique permits augmenting the functionality of a method on an object with "rules" (DeGroot, Abstract). With DeGroot's system a user submits declarative statements with are associated with a method identified on an object such that when the method on the object is called the declarative statements are executed in addition to the methods on the object (DeGroot, Abstract). DeGroot's "rules" are high level requirements, parameters or constraints that are described as follows:

> In general, rules are high level requirements or functions that augment the functionality of methods of an object. In one embodiment, rules are declarative techniques, that set forth requirements or parameters that define object behavior.
>
> (DeGroot, column 4, lines 40-41)

DeGroot's rules, may, for example, be used to define the range of parameters acceptable for execution of a method on an object. For example, a rule for a "time" method may check whether the value stored in the "day" attribute is a valid day for the corresponding "month" parameter (e.g., the day parameter is a value between 1 and 30 for the month of September) (DeGroot, col. 5, lines 32-40). DeGroot's rules may also be used to set requirements prior to invocation of a method. For example, in an order entry application program, the user may specify the requirement that before the "ship" method is executed, the shipping information, used by the ship method, must exist (DeGroot, col. 5, lines 41-48). As illustrated by these examples, the rules described by DeGroot comprise constraints or requirements associated with a given method of an object (also see, e.g., DeGroot's claim 1). There is a fundamental difference between the "rules" defined by DeGroot et al. and the "attributes" of Applicant's invention.

Applicant's invention provides for use of attributes as a flexible and extensible way of modifying the behavior of a program (or a class or method of a program). As described below in more detail, the term "attributes" refers to a language construct that developers (users) can use to add metadata to code elements (e.g., assemblies, modules, members, types, return values, and parameters) to extend their functionality (see e.g., Applicant's specification, paragraph [0031]). A developer (user) can utilize Applicant's attribute-based component programming system to define attributes that specify declaratively the addition of certain behavior to a program being developed. These <u>attributes comprise "active" metadata used to generate code for adding behavior to a program as it executes at runtime</u>.

Applicant's attribute-based component programming system and methodology provides a flexible and extensible way of modifying the behavior of a program or a class or a method of a program. Applicant's attributes are used to generate additional code for inclusion in a program class or subclass. For instance, a developer may define attributes that specify the addition of method tracing behavior to a class being developed in order to assist in debugging the program that is being developed (see e.g., Applicant's specification, paragraph [0080]). This tracing behavior may, for instance, cause a message to be displayed on the screen indicating that a particular method was called, specifying the calling parameters with which the method was called, and providing the result returned by the method (see e.g., Applicant's specification, paragraph [0083]). <u>Attributes that are defined and added to a class or method result in additional code being automatically generated to implement the additional behavior (e.g., method tracing behavior) when the method is called at runtime</u> (see e.g., Applicant's specification, paragraph [0084]). For example, a developer may add the above-described tracing attributes to an “add” method of a given class, so that when the “add” method is called, the defined attribute causes the tracing behavior to be added. Thus, the “attributes” of Applicant’s invention are substantially different from DeGroot's “rules” that define requirements or constraints.

Applicant has amended its claims to bring these distinctions to the forefront. For example, Applicant’s amended claim 1 includes the following claim limitations:

> A method for dynamically generating program code adding behavior to a program based on attributes, the method comprising:
> adding a static field of type Component to a program class of the program to create a component;
> defining at least one attribute specifying declaratively behavior to be added to the program, wherein said at least one attribute comprises active metadata used to generate program code for inclusion in the program;
> associating said at least one attribute with the component; and
> in response to instantiation of the component at runtime, generating a subclass based on the program class and said at least one attribute, the subclass including dynamically generated program code based on said at least one attribute
>
> (Applicant’s amended claim 1, emphasis added)

Additionally, Applicant’s claimed invention also provides for associating the defined attributes with a component so as to generate additional code that adds desired behavior to the program. This is described, for example, in Applicant’s specification as follows:

> ... The attributes added to the class typically result in the generation of extra code that is added into the dynamically generated subclasses. The Component class generates the subclass (i.e., creates the shell) for each of the components (i.e., component objects) that are instantiated. For each of the component objects in the class, the Component class essentially asks the attributes to specify the extra code that is to be inserted into the appropriate methods of the class”
>
> (Applicant’s specification, paragraph [0085], emphasis added).

The Examiner references DeGroot at column 4, lines 1-5 for the corresponding teaching; however the referenced portion of DeGroot provides as follows:

> In response, the object oriented software environment associates the declarative statements to the method identified on the object such that when the method on

the object is called, the declarative statements, associated with the object, are executed in addition to the methods on the object.

(DeGroot, col. 4, lines 1-5)

As illustrated above, DeGroot's solution does not generate additional code to extend program functionality. Instead, declarative statements written by the developer are called in addition to the methods on the object. Additionally, in DeGroot's system associating declarative statements (or rules) with a method involves redirecting a call to the method to a "pinch-point" operation (see e.g., DeGroot col. 11, lines 55-63). The "pinch-point" operation determines whether there are any rules associated with the method called (DeGroot, col. 5, lines 14-21). If rules are determined to exist, then one or declarative statements associated with the method are called (DeGroot, col. 5, lines 21-29; see also, claim 8).

In contrast, Applicant has no equivalent notion of "pinch-point"; nothing is dynamically determined when a method having associated attributes is called. So although at a high level DeGroot describes a "declarative" system, it could more accurately be described as being based on "dynamic rule evaluation leading to declarative statement execution". This differs dramatically from Applicant's invention which generates subclass code based on active metadata (i.e., attributes).

This introduces another distinction between Applicant's invention and DeGroot's solution. In response to instantiation of a component at runtime, Applicant's invention generates a subclass based on the program class and the defined attribute. The subclass includes dynamically generated program code based on the defined attribute. Here it should be noted that DeGroot in fact appears to teach away from Applicant's invention as DeGroot rejects the notion of sub-classing as a viable basis for their system. The portion of DeGroot referenced by the Examiner describes the usage of subclassing in prior art solutions (DeGroot, col. 2, lines 40-42). However, DeGroot notes that subclassing technique requires re-compiling the code, which in turn requires shutting down the application and reloading both the new and old code. Thus, DeGroot looks for another solution, stating "it is desirable to provide a technique to permit augmentation of a method, to alter the behavior of an object, even though the source metadata is not available…." (DeGroot, col. 2, lines 43-46). DeGroot proceeds to then describe a system which is implemented without use of subclassing.

While DeGroot rejects subclassing, Applicant’s invention utilizes subclassing in a novel way. Applicant’s invention's provides a technique for subclass generation via active metadata (attributes) which is not taught or suggested by DeGroot. Applicant’s invention generates a subclass based on the original class and the attributes defined for the class. The attributes added to the class result in the generation of extra code that is added into the dynamically generated subclasses as previously described (see e.g., Applicant’s specification, paragraph [0085]).

Additional distinctions may also be found in Applicant’s dependent claims. For instance, Applicant’s claim 2 includes claim limitations of defining a particular attribute using active metadata, so as to provide a mechanism for generation of program code from said particular attribute. The portion of DeGroot referenced by the Examiner for the corresponding teaching provides as follows: “To implement subclassing, a program developer requires access to the object’s metadata to modify the object’s v_table” (DeGroot, col. 2, lines 26-28). Although DeGroot mentions metadata, there is no mention of using active metadata for generation of program code in the manner provided in Applicant’s specification and claims.

All told, DeGroot provides no teaching of defining attributes comprising active metadata used to generate code for adding behavior to a program as it executes at runtime. Additionally, DeGroot rejects the notion of subclassing while Applicant’s invention's provides for subclass generation via active metadata (attributes). Therefore, as DeGroot does not teach or suggest all of the claim limitations of Applicant's claims 1-5, 15-19, 21-26, 28-30, 33, 40-43, and 45-46 (and other claims) it is respectfully submitted that the claims distinguish over this reference and overcome any rejection under Section 102.

B. Section 103 rejection: DeGroot and Foster

Claims 6-14, 20, 31-39, and 44 stand rejected under 35 U.S.C. 103(a) as being unpatentable over DeGroot (above) in view of U.S. Patent 7,103,885 of Foster (hereinafter “Foster”). Here, the Examiner acknowledges that DeGroot does not teach various aspects of these claims including, for example, defining attributes based on comments in source code of a program class (Applicant's claim 6), defining attributes in a property file external to the program class and compiling the class containing dynamic attributes from the property file (Applicant's claims 10 and 11) and various other limitations of claims 6-14, 20, 31-39, and 44.

Under Section 103(a), a patent may not be obtained if the differences between the subject matter sought to be patented and the prior art are such that the subject matter as a whole would have been obvious at the time the invention was made to a person having ordinary skill in the art to which the subject matter pertains. To establish a prima facie case of obviousness under this section, the Examiner must establish: (1) that there is some suggestion or motivation, either in the references themselves or in the knowledge generally available to one of ordinary skill in the art, to modify the reference or to combine reference teachings, (2) that there is a reasonable expectation of success, and (3) that the prior art reference (or references when combined) must teach or suggest all the claim limitations. (See e.g., MPEP 2142). As will be shown, the DeGroot and Foster references cited by the Examiner, even when combined, fail to meet the requisite condition of teaching or suggesting all of Applicant's claim limitations.

Applicant's claims are believed to be allowable for at least the reasons cited above (as to the Section 102 rejection) pertaining to the deficiencies of DeGroot as to Applicant's claimed invention. As these claims are dependent upon, and incorporate the limitations of Applicant's independent claims, they are distinguishable for the reasons previously described in detail. As Foster does not provide any teaching of defining attributes comprising active metadata used to generate code for adding behavior to a program as it executes at runtime, it does not cure the deficiencies of the primary DeGroot reference as to Applicant's invention.

Foster's solution comprises a source code version management system. Although Foster does discuss "attributes" associated with source code files, the attributes described by Foster are represented by the presence (or not) of an attribute tag and an associated value in the comment field of a version management file (Foster col. 4, lines 10-13). For example, the software module may be a data file and one attribute may be whether or not the data file may be translated (Foster col. 4, lines 17-19). The software module is then processed based on whether a predetermined tag is found in the comment field of the version management file and, if it is found, the value associated with the tag (Foster col. 4, lines 23-26). For example, if no tag is found then the file is not translated (Foster col. 5, lines 1-5). Clearly, the attributes described by Foster (as well as the solution environment in which they are utilized) are dramatically different than the attributes of Applicant's invention discussed in detail above. Furthermore, Applicant's review of Foster finds no teaching of using the attributes as the basis for generating code for adding behavior to a software program as it executes at runtime.

All told, neither DeGroot nor Foster provide for defining attributes which comprise active metadata and using such attributes for generating code for extending the functionality of a software program. Accordingly, as the combined references do not teach all of the limitation of Applicant's claims, it is respectfully submitted that the claims distinguish over the combined references and overcome any rejection under Section 103.

Any dependent claims not explicitly discussed are believed to be allowable by virtue of dependency from Applicant's independent claims, as discussed in detail above.

Conclusion

In view of the foregoing remarks and the amendment to the claims, it is believed that all claims are now in condition for allowance. Hence, it is respectfully requested that the application be passed to issue at an early date.

If for any reason the Examiner feels that a telephone conference would in any way expedite prosecution of the subject application, the Examiner is invited to telephone the undersigned at 925 465 0361.

Respectfully submitted,

Date: June 30, 2008

/G. Mack Riddle/

G. Mack Riddle, Reg. No. 55,572
Attorney of Record

925 465-0361
925 465-8143 FAX